An Ontology for Describing ETL Patterns Behavior

Bruno Oliveira[1] and Orlando Belo[2]
[1]CIICESI, School of Management and Technology, Porto Polytechnic, Felgueiras, Portugal
[2]ALGORITMI R&D Centre, University of Minho, Campus de Gualtar, 4710-057 Braga, Portugal

Keywords: Data Warehousing Systems, ETL Conceptual Modeling, ETL Patterns, Domain Specific Language, Ontologies, PL4ETL, ETL Skeletons.

Abstract: The use of software patterns is a common practice in software design, providing reusable solutions for recurring problems. Patterns represent a general skeleton used to solve common problems, providing a way to share regular practices and reduce the resources needed for implementing software systems. Data warehousing populating processes are a very particular type of software used to migrate data from one or more data sources to a specific data schema used to support decision support activities. The quality of such processes should be guarantee. Otherwise, the final system will deal with data inconsistencies and errors, compromising its suitability to support strategic business decisions. To minimize such problems, we propose a pattern-oriented approach to support ETL lifecycle, from conceptual representation to its execution primitives using a specific commercial tool. An ontology-based meta model it was designed and used for describing patterns internal specification and providing the means to support and enable its configuration and instantiation using a domain specific language.

1 INTRODUCTION

Ontologies are being used by many organizations to encode and share information across multiple systems, providing a way to electronic agents understand and use the information based on a solid and shared formalism. The need to reuse a particular domain knowledge is growing, since it enhances better solutions and provides a better picture of a specific domain (Gruber 1993). The struggle imposed by global market demands affects business requirements in an unexpected way. Therefore, software design techniques should guaranty the quality and robustness of any software piece. The use of software patterns is a reuse-based technique often applied in software developing on a lot of different domains (Gamma et al. 1995). The need to reuse components and share acquired knowledge across applications is crucial to reduce time and costs of developing software, contributing to improve the quality of the software (Alexander et al. 1977).

In the field of *Data Warehousing Systems* (DWS), the ETL (Extract, Transform, and Load) process is one of the most important pieces that support the entire business intelligence system, consuming a large portion of time and resources in its development. ETL processes are very particular, being specific to each scenario where they are applied, since its main purpose is to integrate data from different data sources to target repositories, which are especially built to support decision-making processes. The amount of data that is typically transformed associated to data requirements and technology limitations that should be considered in its development, places these software systems in a very special domain (Weske et al. 2004). All this contributes for increasing the complexity related to its development and maintenance. Additionally, there is still a lack of proposals and methodologies to support its development based on a conceptual approach with the ability to represent all operational stages with a simple notation and provide at the same time the necessary bridges to allow for its mapping into a correspondent physical model. Based on these problems, we propose a pattern-based approach designed to map typical ETL standard tasks - e.g., *Surrogate Key Pipelining* (SKP), *Slowly Changing Dimensions* (SCD), and *Change Data Capture* (CDC) - to configurable components that can be adapted to specific application scenarios.

Oliveira, B. and Belo, O.
An Ontology for Describing ETL Patterns Behavior.
DOI: 10.5220/0005974001020109
In *Proceedings of the 5th International Conference on Data Management Technologies and Applications (DATA 2016)*, pages 102-109
ISBN: 978-989-758-193-9

Based on previous works (Oliveira & Belo 2012; Oliveira & Belo 2013), and using the Web Ontology Language (OWL) (McGuinness & van Harmelen 2004), an ETL pattern based ontology was developed to support the necessary requirements and to describe each pattern configuration, enabling its mapping to physical models that can be executed in practice (Oliveira & Belo 2015). Basically, an intermediate layer is provided to separate technical knowledge, typically used in commercial tools, from the domain knowledge used by decision-makers (McGuinness & Wright 1998). Due the complexity of the knowledge involved and the application of each pattern to specific contexts (Dietrich & Elgar 2007; Noy & McGuinness 2001), ETL processes can suffer from inconsistencies and misunderstandings about communication problems that can result from different meanings or architectural contradictions. Ontologies can be used to provide the contextual data necessary to describe each pattern according to its structural properties (Noy & McGuinness 2001).

Thus, after a brief exposure of some related work (Section 2), we describe our ontology approach to support ETL patterns, providing a specific taxonomy of the most used ETL techniques and the main components that support the configuration of each pattern (Section 3). Next, a set of necessary formalisms to create a pattern-based language and how to use them to generate physical models is presented (Section 4). Finally, we discuss the experiments done so far, analyzing results and presenting some conclusions and future work (Section 5).

2 RELATED WORK

The development of more abstract models to support the development of ETL processes and their mapping to execution primitives is not new. Vassiliadis and Simitsis covered several aspects of ETL development in their research (Vassiliadis et al. 2003). They approached ETL conceptual modeling (Vassiliadis et al. 2002a), its representation using logical views (Vassiliadis et al. 2002b; Simitsis & Vassiliadis 2008), and its implementation using a specific ETL tool (Vassiliadis et al. 2000). Akkaoui (Akkaoui & Zimanyi 2009) proposed a conceptual approach for ETL development based on well-known technologies such as BPMN (*Business Process Model and Notation*) and BPEL (*Business Process Execution Language*). Several mappings rules were presented to support the mapping of BPMN models to BPEL executable models. This is not easy to make, suffering this approach from several traditional problems already debated by research community (White & Corp 2005). Later, Akkaoui presented the BPMN4ETL meta model (Akkaoui et al. 2011), showing how BPMN conceptual primitives can be mapped to physical models using specific templates recognized by commercial tools. However, in the field of ETL patterns, there is not much more to refer. Even so, we can refer also the work of Köppen et al. (2011), which presented a pattern-oriented approach to support ETL development, providing a general a description for a set of patterns - e.g., aggregator and duplicate elimination patterns. This work was focused on important aspects related to the definition of internal composition properties of patterns and their relationships. The patterns were presented only at a conceptual level, lacking to support patterns instantiation for execution primitives. Thus, we believe that our work distinguishes from other approaches presented so far, since we followed a pattern-based approach supported by well-documented components that can be configured and used in different ETL development phases. Fine-grained tasks are encapsulated inside these components, resulting in a coarse-grained new ETL development level, defined by the use of an upper abstraction layer that simplifies and carries the acquired knowledge between projects.

3 ETL META MODEL FOR PATTERNS DEFINITON

Nowadays, sharing and reusing knowledge it is a crucial activity for software development. Many specific frameworks appeared with the goal to define a new kind of software programming for taking advantages of previous expertise and allowing for its reuse on new applications in different application scenarios and domains. Usually, these frameworks are composed by collections of software patterns representing a set of instructions or activities, which can be configured and applied to more specific needs. Concerning the specificities of an ETL environment, patterns can be characterized using a set of pre-established tasks grouped based on a specific configuration related to the context in which are used. Creating these reconfigurable components avoid the need to rewrite some of the most repetitive tasks that are used regularly. Several tasks, such as surrogate key process generation, lookup operations, data aggregation, data quality filters or slowly changing dimensions, are just some few examples of

usual tasks used in any DWS. Instead of using repetitive tasks to solve the same problems over and over again, conceptual models can be used to simplify ETL representation. This way, users focus on more general requirements, leaving the complexity of its implementation on other development steps. Consequently, users only need to provide configuration metadata to the conversion engine that will be responsible to generate the correspondent physical model.

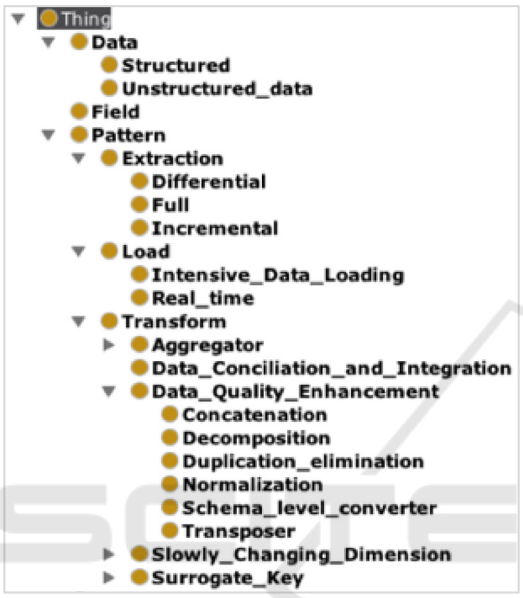

Figure 1: The ETL patterns taxonomy.

OWL, a language based on Web semantic technology, is often used to describe domain specific meta-models in order to represent properties and relationships between domain concepts (i.e., patterns). OWL is a W3C standard (W3.org 2012) that was developed to provide a simple form to process and use semantic data across applications in the Web. With OWL, classes or concepts can be described and arranged to form taxonomic hierarchies, properties describing the composition in terms of attributes of each concept and restrictions over the relationship between the concepts presented. Thus, ETL patterns can be syntactically expressed using classes, data properties and object properties, providing the basic structure to support the development of a specific language to pattern instantiation. Figure 1 shows an excerpt of the breakdown among the different levels of the ETL patterns taxonomy proposed. The 'Pattern' class represents the most general concept used, while 'Extraction', 'Transform' and 'Load' are the three types of patterns that are intrinsically associated to each typical phase of an ETL process. Instances of 'Extraction' are used to extract data from information system using a specific data object (e.g., a table or file), representing typical data extraction processes and algorithms applied over specific data structures. Three types instances are commonly referred for the concept 'Extraction', namely:

a) Full extraction patterns that are used to extract all data from a specific data source without any criteria, i.e., all data currently available.
b) Differential extraction patterns that are used to identify new data since the last successful extraction. For this data extraction type, all data from source and target repository is compared to identify new data.
c) Incremental extraction patterns: used to extract data from data sources since the last successful extraction but based on specific criteria and using specific CDC (*Change Data Capture*) techniques to identify and track the data that has been changed in all the data warehouse sources.

The 'Transformation' class represents patterns that are used in ETL transformation phase for the application of a set of cleaning or conforming tasks (Rahm & Do 2000), in order to align source data structures to the requirements of the target schema of a data warehouse. This class represents a large variety of procedures that are often applied in DWS, such as patterns responsible to apply the well-known policies related to SCD techniques, patterns for surrogate key generation, or patterns to support the conciliation and integration of data from many data sources. For example, a DQE pattern can be specialized to a 'Normalization' class, which represents the set of tasks needed whenever it is necessary to standardize or correct data according to a given set of mapping rules stored in mapping tables. With these classes, all the most frequent ETL patterns can be represented along with all its operational stages. Using the ontology hierarchy to support ETL patterns meta-model, patterns can be changed or even new patterns can be added without compromising the whole pattern structure. Finally, The 'Load' class represents patterns that are used to load data to the target DW repository, representing efficient algorithms for data loading or index creation and maintenance for loading procedures. The '*Intensive Data Loading'* (IDL) subclass should load data to a target DW schema considering the model restrictions used.

After the taxonomy definition, the meta-model should be enriched to support the basic rules for the development of well-formed ETL patterns. For that, each class should be defined through the use of properties. For example, the 'Extraction' class representing all 'Extraction' patterns is composed by some *Datatype* Properties such as *PatternId* and

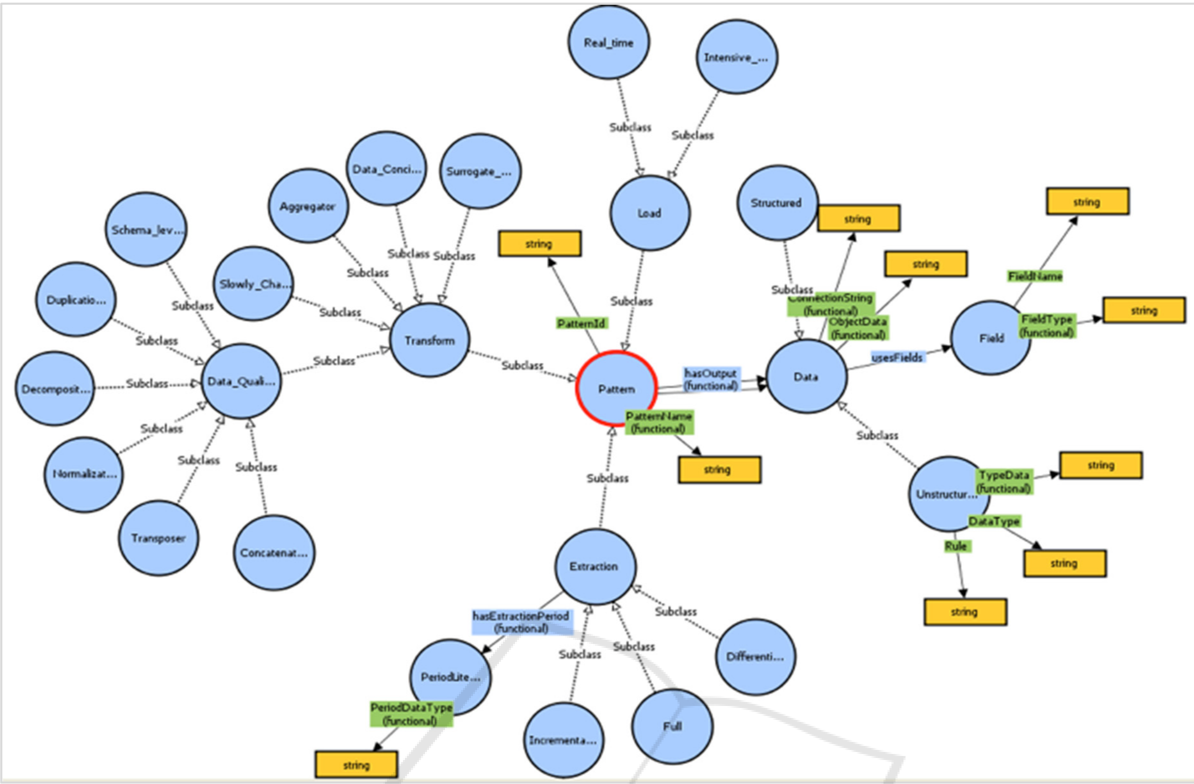

Figure 2: The ontology for ETL patterns – a view.

PatternName (inherited from 'Pattern' class), and Object Properties such as *PeriodLiterals* that refers to extraction interval used (Hour, Daily, Month) (*oneOf* property) and the metadata related to the repository connection (input object: Data class and fields used: Field class). Each subclass can also include additional properties. For example, the 'Incremental' class uses a date type property to identify new or changed records. Each property should be described based on its cardinality, value, domain and range. The domain links a property to a class, while the range links a property to a class or data range. This allows for the association between classes and data types, and provides a way to establish restrictions. For example, all patterns should have (*hasInput* object property) at least one (*minCardinality* restriction) and at most two (*maxCardinality* restriction) 'Data' class association for pattern input and only one (*functionalProperty* constraint) 'Data' class association for pattern output. Generally, patterns only have one source for input and one target repository as output. However, the DCI (*Data Conciliation and Integration*) pattern uses more than one data source as input (using *subPropertyOf* axiom) due being responsible to integrate data extracted from two data sources from the same data object. In Figure 2 we can see a brief resume of some classes and its data or object properties.

4 PATTERN LANGUAGE SPECIFICATION

Several authors tried to simplify and minimize ETL development using conceptual models in early development phases. Currently, there is still a lack of semantics to support ETL systems specification and development, and more importantly to provide the necessary mappings to execution properties, taking advantage of the work done previously in design phases. The majority of works presented till now supports ETL processes representation using very detailed tasks. Thus, the models that are generated automatically are composed by dozens of tasks without a direct mapping to commercial tools. With the pattern-based approach proposed, a new abstraction layer is provided, simplifying and helping ETL development from conceptual phases to physical models that can be executed. For this particular task, we believe that commercial tools should be preferentially used, since they provide powerful and well-known frameworks that many

professionals are able to use. Therefore, we propose a specific configuration language that can be applied to each pattern presented by the ontology, covering its operational stages and providing a solid framework to enable its conversion to equivalent semantics used by current ETL commercial tools. Using the Protégé-OWL API (Protégé 2011) (Horridge 2012) we can use and manipulated a specific RDF/XML (Brickley & Guha 2004). Based on the concepts and properties presented, a specific generator was built to automatically generate a specific pattern configuration language, allowing for the configuration of each pattern using the ontology definition. The engine uses two important layers: the language construction rules (syntax), and the ontology data model. For the language specification, a set of type statements and keywords were used to describe each language component. The *USE* keyword is used to identify the pattern path that should be used based of the taxonomy presented (Figure 1), followed by the pattern name. Top levels (*Pattern* class is the higher level) should be firstly defined and the special character '*.*' (dot) is used to traverse each hierarchy level, from the middle levels to bottom levels. Next, and based on each *Pattern* class object properties, a set of blocks delimited by {} (braces) are defined. Inside each block, simple or composite assignments can be performed. For the general blocks (generated from *Pattern* class), the simple assignments are formed based on data properties associated to *Pattern* class, while composite statements are generated based on the object properties. Each block can contain more than one occurrence based on the cardinality of the object properties associated to '*Pattern*' class. For example, the DCI pattern have two input block, each one for the data source used for data integration. The *OPTIONS* block is used to map the properties associated to the bottom pattern class used and can be composed by single or composite statements.

Using the ontology and the syntax rules presented, the configuration language can be automatically generated for each pattern. This approach guarantees that, if the ontology is change, then the correspondent grammar rules will be consistent with the ontology definition. Figure 3 shows an example of the syntax rules applied to the language constructs (Figure 3a) and a correspondent example of its instantiation using a specific pattern (*Aggregator*) that applies a *sum* operation to the duration of telephone calls made by each customer.

The *sum_duration* aggregator pattern (*Transform.Aggregator*) presents three main blocks derived from the object properties applied to the *Pattern* class. The *Source* describes input metadata, *Target* describes output metadata and *Fields* block describes the fields will be used as output to the

```
USE pattern_name
   block_name_1{
        ([)
        single_statement_name
and
        [composite
statement_name{…}]
     ([])(,)(…)
    }
    ' '
```

```
USE Transform.Aggregator
'sum_duration'
   source{
      data=CDR_Calls.csv
      type=CSV{
            delimiter=':'
      }
   }
   target{
     data=calls
      type=relational
   }
   fields{
      DATETIME, CustomerId
   }
```

Figure 3: PL4ETL basic pattern configuration syntax and language example through the instantiation of an aggregator pattern.

target repository. These three blocks correspond to *hasInput*, *hasOutput* and *hasFields* object properties, respectively. For input block, a CSV file was used for data extraction based on delimiter '*:*' (a composite statement is used due the existence of a data property describing the delimiter rule for the *CSV* class), and the pattern output will store correspondent data into a specific relational table. Details such as database name or server was omitted since they can be configured in further steps. After fields identification (separated by comma), the keyword *OPTIONS* is used to specify each configuration parameter (derived from properties applied to the *Aggregator* class) associated to Aggregator class: *Function* to identify the aggregation function applied, *FunctionField* to specify the field that should be used by the function, *RenameField* to apply the alias to the new field generated and the *GroupFields* used to specify the *group by* clause.

With this pattern-based approach, a new abstraction layer for developing ETL processes is proposed. Patterns can be used to create an ETL conceptual model without focusing in very detailed tasks. However, to produce physical models based on conceptual primitives we need to provide two

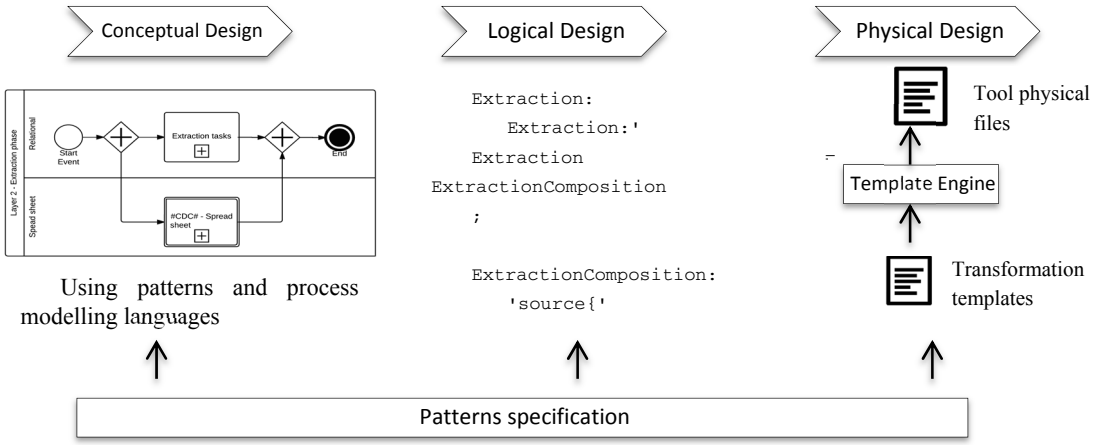

Figure 4: Development stages of an ETL process using patterns.

independent components: patterns configuration meta data that is supported by the domain language provided, and workflow coordination data that describes the process flow. Only for demonstration purposes, the BPMN language was used to create the ETL conceptual models we used. In several works, BPMN has proven that is quite suitable to represent several workflow operational components of ETL systems, both at conceptual and physical primitives (Akkaoui et al. 2012; Oliveira et al. 2015). In recent works (Oliveira et al. 2014; Oliveira & Belo 2015) we also proposed the use of BPMN as visual layer to support ETL conceptual models, representing patterns using BPMN elements. The experimental tool developed has the ability to interpret the configuration language and provide the generation of a physical model, making it possible to be executed by commercial tools such as Kettle from Pentaho (Bouman & Dongen 2009). For that and based on the ontology presented, a specific meta model can be generated and used to support pattern instantiation and configuration. Using protégé editor to create the ontology it's possible to generate java code based on the ontology definition. This feature allows for manipulating the ontology and, at the same time, provides the necessary contracts to control and implement the necessary models to support pattern interpretation and manipulation. The final step covers the generation of physical models using the architecture and philosophy followed by each commercial tool. A set of standard transformation skeletons was built to encapsulate the logic of the conversion process, providing the meanings to transform each pattern internal structure to a specific serialization format. Figure 4 summarizes all the phases of the development process that are needed to support the physical representation of an ETL process using patterns, from the ontology definition to the generation of physical model.

5 CONCLUSIONS AND FUTURE WORK

In practice, ETL systems are sophisticated data migration processes that follow some traditional guidelines associated to repositories that respect to some specific architectural philosophies. The specificities of ETL systems have been studied and applied to several areas, contributing to the identification of common tasks and solutions in order to solve them. The SCD policies are just an example of a typical procedure, identified and categorized based on acquired experience over the years. Additionally, the complexity of the tasks and operators in a complicated workflow consumes great amounts of time and computational resources. With the pattern-oriented approach present in this paper, the knowledge and best practices revealed by several works can be put in practice using a set of software patterns that can be applied to the entire ETL development life cycle: from conceptual phase to its physical implementation primitives. Such patterns can be used in software models, providing a new level of abstraction that simplifies the initial phases of software development, especially the ones related to requirements elicitation and generation of conceptual models.

Patterns provide an excellent groundwork for process validation, allowing for the identification of the most important parts of a system to be built. In order to formalize its composition, we presented an ontology specification describing and categorizing

all the ETL patterns proposed, having the ability to express the construction rules for a language we built previously to support the configuration of each ETL pattern. The ontology also describes the main operational components of each pattern, covering the main properties and restrictions that can be used to support its usage. Thus, enriching each pattern definition using in the referred language, we can use all the main components to its posteriorly the mapping of execution primitives. Recognizing the value and the abilities of the frameworks offered by commercial migration tools, we can develop specific transformation templates to translate each ETL pattern configuration to a corresponding format that can be interpreted directly by an ETL implementation tool. All this provides pattern reusability across several systems and contributes to system robustness, since patterns are independent elements on every ETL applications.

As future work, a set of tests will conducted to study the feasibility of our approach as well as to extend it, improving and enriching the ontology in order to cover more coordination and communication aspects, essentially.

REFERENCES

Akkaoui, Z. et al., 2011. A model-driven framework for ETL process development. *Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP (DOLAP'11)*, pp.45–52.

Akkaoui, Z. et al., 2012. BPMN-Based Conceptual Modeling of ETL Processes. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7448, pp.1–14.

Akkaoui, Z. & Zimanyi, E., 2009. Defining ETL worfklows using BPMN and BPEL. In *Proceeding of the ACM twelfth international workshop on Data warehousing and OLAP DOLAP 09*. pp. 41–48. Available at: http://portal.acm.org/citation.cfm?doid=1651291.1651299.

Alexander, C., Ishikawa, S. & Silverstein, M., 1977. A Pattern Language: Towns, Buildings, Construction, *Oxford University Press*.

Bouman, R. & Dongen, J. Van, 2009. Pentaho® Solutions: Business Intelligence and Data Warehousing with Pentaho and MySQL®,

Brickley, D. & Guha, R.V., 2004. RDF Vocabulary Description Language 1.0: RDF Schema. W3C, pp.1–15. Available at: http://www.w3.org/TR/rdf-schema/.

Dietrich, J. & Elgar, C., 2007. Towards a web of patterns. Web Semantics: Science, Services and Agents on the World Wide Web, 5(2), pp.108–116. Available at: http://linkinghub.elsevier.com/retrieve/pii/S1570826807000030.

Gamma, E. et al., 1995. Design patterns: elements of reusable object-oriented software. *Design, 206*, p.395. Available at: http://www.cs.up.ac.za/cs/aboake/sws780/references/patternstoarchitecture/Gamma-DesignPatternsIntro.pdf.

Gruber, T.R., 1993. A translation approach to portable ontology specifications. *Knowledge Acquisition, 5(2),* pp.199–220. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.101.7493.

Horridge, M., 2012. protégé-owl api. Research, Stanford Center for Biomedical Informatics, p.1. Available at: http://protege.stanford.edu/plugins/owl/api/.

Köppen, V., Brüggemann, B. & Berendt, B., 2011. Designing Data Integration: The ETL Pattern Approach. *The European Journal for the Informatics Professional, XII(3)*.

McGuinness, D.L. & van Harmelen, F., 2004. OWL Web Ontology Language Overview, *OMG*.

McGuinness, D.L. & Wright, J.R., 1998. Conceptual modelling for configuration: A description logic-based approach. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 12(4)*, pp.333–344.

Noy, N. & McGuinness, D., 2001. Ontology development 101: A guide to creating your first ontology. Development, 32, pp.1–25. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.136.5085&rep=rep1&type=pdf\nhttp://liris.cnrs.fr/alain.mille/enseignements/Ecole_Centrale/What is an ontology and why we need it.htm.

Oliveira, B. et al., 2015. Conceptual-physical bridging - From BPMN models to physical implementations on kettle. In *CEUR Workshop Proceedings*. pp. 55–59.

Oliveira, B. & Belo, O., 2015. A Domain-Specific Language for ETL Patterns Specification in Data Warehousing Systems. In *17th Portuguese Conference on Artificial Intelligence.*

Oliveira, B. & Belo, O., 2012. BPMN Patterns for ETL Conceptual Modelling and Validation. In *20th International Symposium on Methodologies for Intelligent Systems*.

Oliveira, B. & Belo, O., 2013. Pattern-based ETL conceptual modelling. In *3rd International Conference on Model & Data Engineering (MEDI 2013)*.

Oliveira, B., Belo, O. & Cuzzocrea, A., 2014. A Pattern-oriented Approach for Supporting ETL Conceptual Modelling and its YAWL-based Implementation. In *4th International Conference on Data Management Technologies and Applications*.

Protégé, 2011. The Protégé Ontology Editor, Available at: http://protege.stanford.edu/.

Rahm, E. & Do, H., 2000. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull., 23(4),* pp.3–13. Available at: http://wwwiti.cs.uni-magdeburg.de/iti_db/lehre/dw/paper/data_cleaning.pdf\npapers2://publication/uuid/17B58056-3A7F-4184-8E8B-0E4D82EFEA1A\nhttp://dc-pubs.dbs.uni-leipzig.de/files/Rahm2000DataCleaningProblemsand.pdf.

Simitsis, A. & Vassiliadis, P., 2008. A method for the mapping of conceptual designs to logical blueprints for ETL processes. *Decision Support Systems, 45(1)*,

pp.22–40.

Vassiliadis, P. et al., 2003. A framework for the design of ETL scenarios. In *Proceedings of the 15th international conference on Advanced information systems engineering. CAiSE'03*. Berlin, Heidelberg: Springer-Verlag, pp. 520–535. Available at: http://dl.acm.org/citation.cfm?id=1758398.1758445.

Vassiliadis, P. et al., 2000. ARKTOS: A tool for data cleaning and transformation in data warehouse environments. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, pp.1–7. Available at: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:A+Tool+For+Data+Cleaning+and+Transformation+in+Data+Warehouse+Environments#0.

Vassiliadis, P., Simitsis, A. & Skiadopoulos, S., 2002a. Conceptual Modeling for ETL Processes. *Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP - DOLAP '02*, pp.1–25. Available at: http://dl.acm.org/citation.cfm?id=583890.583893.

Vassiliadis, P., Simitsis, A. & Skiadopoulos, S., 2002b. *On the Logical Modeling of ETL Processes. Science,* pp.782–786. Available at: http://www.springerlink.com/index/tjaep8rw6y7nrb07.pdf.

W3.org, 2012. Semantic Web - W3C. W3.org. Available at: http://www.w3.org/standards/semanticweb/.

Weske, M., van der Aalst, W. & Verbeek, H., 2004. Advances in Business Process Management. *Data & Knowledge Engineering, 50*, pp.1–8.

White, S.A. & Corp, I.B.M., 2005. Using BPMN to Model a BPEL Process. *Business, 3,* pp.1–18. Available at: http://www.bpmn.org/Documents/Mapping_BPMN_to_BPEL_Example.pdf.